FOAF UI

Käyttöliittymäkerroksen prototyyppi

Janne Ylinen

Opinnäytetyö
Toukokuu 2011

Mediatekniikka
Tekniikan ja liikenteen ala

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖN
KUVAILULEHTI

Tekijä(t)	Julkaisun laji	Päivämäärä
YLINEN, Janne	Opinnäytetyö	12.05.2011
	Sivumäärä 30	Julkaisun kieli Suomi
	Luottamuksellisuus ( ) saakka	Verkkojulkaisulupa myönnetty (X)

Työn nimi
FOAF UI - Käyttöliittymäkerroksen prototyyppi

Koulutusohjelma
Mediatekniikan koulutusohjelma

Työn ohjaaja(t)
MANNINEN, Pasi

Toimeksiantaja(t)
SUNI, Juha. Media Cabinet Oy

Tiivistelmä
WWW-sisällönhallintajärjestelmiä on kritisoitu kankeaksi ja rajoittavaksi tekijäksi nykyaikaisissa WWW-projekteissa. Ratkaisuna tähän pidetään karkeamman tason arkkitehtuuria, minkä ansiosta järjestelmä on räätälöitävissä vapaammin. Sovellusalustan tarkoituksena on tarjota pieniä ja yleiskäyttöisiä moduuleita sovelluksen rakentamiseen. Tähän strategiaan perustuen Media Cabinet Oy on lähtenyt kehittämään täysin uutta tuotetta perinteisen sisällönhallintajärjestelmänsä rinnalle.

Opinnäytetyössä suunniteltiin ja kehitettiin prototyyppi FOAF:ksi nimetyn uuden järjestelmän käyttöliittymäkerroksesta. Tavoitteena oli löytää työkalut ja tekniikat määritellyn toiminnallisuuden toteuttamiseksi. Toteutusvaiheen aikana tehdyn muutoksen jälkeen käyttöliittymäkerros perustui pelkästään selainpuolen teknologioihin. Arkkitehtuurin pohjaksi valittiin JavaScript-kielellä toteutettu jQuery-kirjasto, jonka ominaisuuksiin käyttöliittymäkerroksen DOM-rakenteen manipulointi pohjautuu.

Merkittävimmän osan käyttöliittymäkerroksesta muodostivat jQuery:ä laajentavat lisäosat. jQuery UI:n Widget Factory toiminnallisuutta käyttämällä lisäosien oli mahdollista laajentaa toisiaan perinteisistä olio-ohjelmointikielistä tutulla tavalla. Arkkitehtuuria havainnollistamaan toteutettiin muutamia keskenään hyvin erilaisia komponentteja. Tärkeimpänä näistä Panel- ja Grid-komponentit, joiden avulla esiteltiin yksinkertainen tekniikka luoda käyttöliittymä. Panel-komponenttien tarkoituksena oli toimia ikkunoina, joihin sisältö voidaan hakea esimerkiksi asynkronisesti tietyn väliajoin. Käyttämällä Grid-komponenttia yhdessä Panelien kanssa, oli mahdollista asemoida niistä monimutkainenkin palstarakenne.

Käyttöliittymän prototyypin valmistuttua siihen valitut tekniikat todettiin asiakkaan toimesta käyttökelpoiseksi. Myös osa arkkitehtuurista ja toteutetuista komponenteista otettiin varsinaisen FOAF:n kehitysversion käyttöön. Opinnäytetyössä toteutettujen komponenttien ja koko käyttöliittymän jatkokehitys tulee tapahtumaan asiakkaan toimesta.

Avainsanat (asiasanat)

jQuery, JavaScript, CSS, HTML, Ajax

Muut tiedot

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

DESCRIPTION

Author(s)	Type of publication	Date
YLINEN, Janne	Bachelor´s Thesis	12.5.2011
	Pages 30	Language Finnish
	Confidential ( ) Until	Permission for web publication (X)

Title
FOAF UI – User interface prototype

Degree Programme
Media Engineering

Tutor(s)
MANNINEN, Pasi

Assigned by
SUNI, Juha. Media Cabinet Oy

Abstract
Web content management systems have been recently criticized for being stiff and restrictive factors in modern web projects. As a result more coarse architecture is considered, which allows better customization of the system. The function of an Application framework is to provide small and generic modules for the development of applications. To fulfill this demand, Media Cabinet Oy has started to develop a completely new product in addition to their traditional content management system.

In this bachelor´s thesis a prototype of the user interface layer was designed and developed. The main objective was to research of the proper tools and technologies for developing the defined functionality. Decisions and changes made during development phase resulted in an architecture completely based on browser side functionality. jQuery JavaScript-library was chosen to be the foundation of the user interface. The main criteria for the use for jQuery were its capabilities for DOM manipulation.

The main part of the user interface consists of plugins extending jQuery-library. Using jQuery UI Widget Factory feature plugins enabled to extend each others in a way familiar from traditional object-oriented languages. To demonstrate the architecture few components were developed. The most important of these were Panel and Grid components, which could be used to easily create the user interface for the application. The panels are used as containers, to which the content can be loaded asynchronously with defined time interval. Using Grid component with Panels, they can be positioned to fairly complex columns.

When finished the user interface prototype was reviewed by the customer and the technologies were stated as usable. A major part of the architecture and developed components were deployed to the actual development version of the FOAF application framework. The upcoming development of the components and the whole user interface will be maintained by the customer.

Keywords

jQuery, javaScript, CSS, HTML, Ajax

Miscellaneous

SISÄLTÖ

KUVIOT

1 TYÖN LÄHTÖKOHDAT

1.1 Toimeksiantajan kuvaus

Media Cabinet Oy on Jyväskyläläinen yritys, jolla on toimipiste myös Lahdessa. Yrityksen päätuotteena on pitkän kehityskaaren omaava Base-sisällönhallintajärjestelmä. Base on kehitetty mahdollisimman modulaariseksi, jonka ansiosta sitä voitaan räätälöidä erilaisiin käyttötarkoituksiin sopivaksi.

1.2 Opinnäytetyön tavoitteet ja rajaus

Opinnäytetyön tavoitteena oli rakentaa asiakkaan uuden verkkosovelluksen käyttöliittymäkerroksen prototyyppi. Työnimellä ”FOAF” kulkevasta sovelluksesta tullaan julkaisemaan uusia versioita aktiivisesti ja opinnäytetyön tulos on tarkoitus olla sen käytössä alusta asti. Käyttöliittymäkerrosta jatkokehitetään kuitenkin opinnäytetyön jälkeen, joten valmista tuotetta siitä ei ollut tarkoitus kuitenkaan tulla. Tärkein tavoite oli tutkia erilaisten tekniikoiden soveltuvuutta suunnitellun toiminnallisuuden aikaansaamiseksi.

Toteutettava käyttöliittymäkerros tulee käyttöön sovelluksen ylläpitopuolelle, eikä siis verkkosivuston niin kutsutuille loppukäyttäjille. Tarkoitus oli luoda työkalut ja käytänteet, joiden avulla moduulien käyttöliittymän kehitys on mahdollisimman helppoa ja yhtenäistä. Moduuleita kehittävät ulkopuoliset yritykset, joille tarjotaan avoimet rajapinnat tähän.

Käyttöliittymäkerros perustuu pelkästään selaimen toiminnallisuuteen, eikä siksi periaatteessa tarvitsekaan palvelinta toimiakseen. Opinnäytetyössä ei siis käytetä PHP:tä tai muita palvelinohjelmointikieliä.

1.3 WWW-sisällönhallintajärjestelmät

1.3.1 Yleisesti

WWW-sisällönhalliintajärjestelmillä tarkoitetaan yleisesti sovelluksia, joilla esimerkiksi julkisia verkkosivuja keskitetysti ylläpidetään. Kaiken ytimenä on ajatus sisällön erottamisesta ulkoasustaan ja näin mahdollistaa sen tehokas käyttö. Keskeinen ominaisuus Internet-sivujen ylläpitoon tarkoitetuissa sisällönhallintajärjestelmissä on niin

kutsutut sivupohjat. Sivupohjat toimivat runkona, johon sisällön osia haetaan dynaamisesti.

Tyypillinen esimerkki sisällönhallintajärjestelmän ominaisuudesta ja tarpeesta ovat sisältösivut. Sivujen sisältö saattaa sisältää videoita, kuvia, listoja tai leipätekstiä, joita kaikkia voidaan ylläpitää helposti sisällönhallintajärjestelmästä.

Sisällönhallintajärjestelmissä on hyvin paljon vaihtoehtoja erilaisille tekniikoille ja ympäristöille. Iso osa näistä järjestelmistä on maksullisia palvelinympäristöjensä tai laajuutensa takia. Myös avoimen lähdekoodin vaihtoehtoja on saatavilla monille erilaisille tekniikoille.

1.3.2 Tulevaisuus

Viime aikoina WWW-sisällönhallintajärjestelmiä on kritisoitu vanhanaikaisiksi. Syynä tähän pidetään nykyisten WWW-sivustojen tapaa koota sisältönsä suuresta määrästä lähteitä. Näissä tilanteissa valmista sisällönhallintajärjestelmää pidetään kankeana ja rajoittavana tekijänä. Uusien ominaisuuksien kehittäminen suureen järjestelmään tarpeen tullen on usein raskasta ja tällöinkin huomioonotettavia asioita on paljon.

Ratkaisuna tähän pidetään karkeampaa ja abstraktimpaa järjestelmää. Järjestelmä sisältäisi pieniin komponentteihin purettuja ominaisuuksia ja toimisi kirjastona tai ohjelmistokehyksenä. Näistä komponenteista olisi sitten helppo luoda projekti ilman valmiin järjestelmän rajoituksia. (Tolvanen P. Onko julkaisujärjestelmien aika ohitse? Vastaus: Ei.)

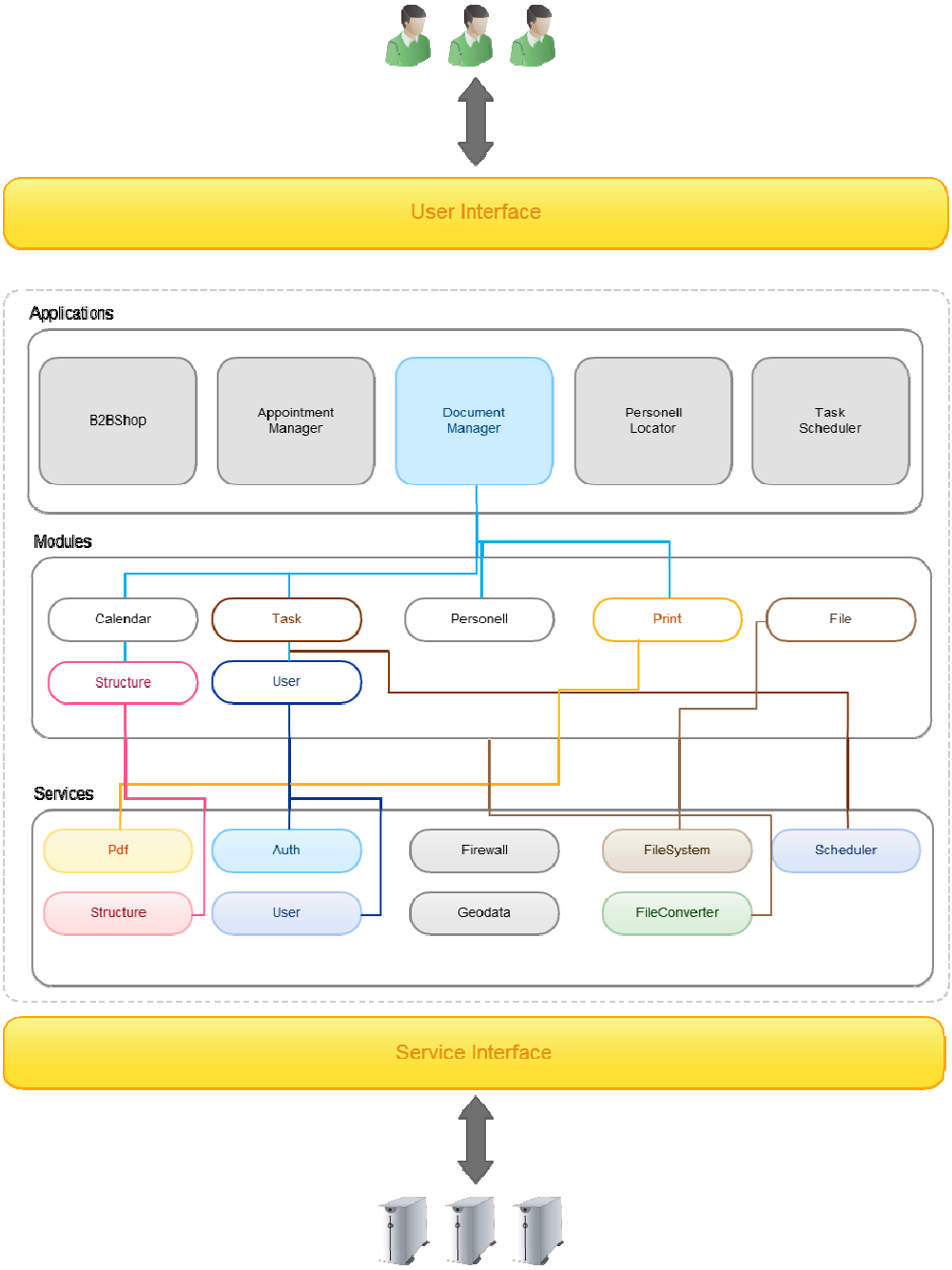
Media Cabinetin tärkein tuote on Base-sisällönhallintajärjestelmä ja aihe on siksi yritykselle ajankohtainen ja tärkeä. Myös Media Cabinetilla tunnetaan ongelmatilanteet, joissa järjestelmä rajoittaa projektin toteutusta.

Media Cabinet on lähtenyt korjaamaan tätä ongelmaa kehittämällä täysin uutta järjestelmää. FOAF ei ole sisällönhallintajärjestelmä, mutta sen avulla voi esimerkiksi sellaisen toteuttaa.

2 FOAF-SOVELLUSALUSTA

2.1 Esittely

FOAF (Finnish Open Application Framework) on sovellusalusta, jonka pohjalta voidaan rakentaa erilaisia Internet-selaimessa toimivia sovelluksia. Sovellusalustan tarkoituksena on tarjota suuri valikoima moduuleita ja käytänteitä sovelluksen rakentamiseen (Ks. kuvio 1.).



KUVIO 1. FOAF-sovellusalustan arkkitehtuuri

Sovellusalusta pohjautuu ideaan avoimen lähdekoodin ja avoimen datan järjestelmästä, joka on arkkitehtuuriltaan vahvasti palvelukeskeinen. Avoin data viittaa yhdessä palvelukeskeisen arkkitehtuurin kanssa siihen sovellusalustan erityispiirteeseen, että kaikki sovelluksien tarjoamat palvelut ovat kutsuttavissa verkon yli mistä tahansa kolmannen osapuolen järjestelmästä.

2.2 Käyttöliittymäkerros

FOAF:n käyttöliittymäkerros vastaa kaikesta, mitä loppukäyttäjä näkee järjestelmän ylläpidossa. Opinnäytetyössä keskitytään tarjoamaan työkalut ja käytänteet moduulikehittäjille luoda muun järjestelmän kanssa yhdenmukaisia käyttöliittymiä.

Tavoitteena oli luoda esimerkkikomponentteja, joita moduulikehittäjät voivat käyttää moduuliensa käyttöliittymässä. Esimerkkielementtien lisäksi haluttiin kehittää yksinkertainen, mutta monipuolinen tapa rakentaa käyttöliittymän asettelu. Kehittäjä pystyisi yksinkertaisilla määrityksillä asemoimaan elementtejä näkymässä.

Yksi merkittävä ominaisuus käyttöliittymässä on sen tuki mobiililaitteille. Käytettäessä esimerkiksi matkapuhelinta käyttöliittymän tulisi tarjota sama sisältö, mutta pienemmälläkin näytöllä käytettävästi.

3 KEHITYSYMPÄRISTÖ

Opinnäytetyön kehitystyöhön haluttiin koota moderneja työkaluja avuksi. Kehitystyötä tultaisiin tekemään usealla eri työpisteellä, joka osaltaan edellytti yhtenäistä toimintatapaa. Tärkeä tekijä ohjelmistojen ja valmiiden komponenttien valinnassa on FOAF:n jatko ja varsinainen kehitysvaihe. Tarkoitus oli testata ja etsiä toimivia ratkaisuja jo nyt ja ottaa niitä käyttöön osaksi käyttöliittymäkerrosta ja kehitysympäristöä.

3.1 Zend Studio

Media Cabinet Oy käyttää laajasti Zend Technologiesien ratkaisuja tuotannossa ja kehitystyössä. HTTP-palvelimena toimii Zend Server ja merkittävä osa PHP-toteutuksista käyttää Zend Framework -ohjelmistokehystä. Myös työpöytäkäytössä on Zendin Eclipse-pohjainen integroitu ohjelmointiympäristö Zend Studio. Täten myös FOAF:n palvelinarkkitehtuuri rakentuu edellä mainittujen tuotteiden pohjalta.

Maksullisena ohjelmistona Zend Studio tarjoaa kehittyneitä ominaisuuksia PHP-kehitykseen, joita ei Eclipsen ilmaisversioista löydy. Zend Studiossa on sisäänrakennettu muuan muassa edistynyt tuki palvelimella sijaitsevien tiedostojen muokkaukseen ja PHP-koodin dokumentointiin ja generointiin.

Näiden lisäksi Zend Studio tunnistaa työasemalle asennetun Zend Serverin ja Zend Frameworkin, sekä osaa hyödyntää niitä. Projekti voidaan luoda suoraan asennettuun paikalliseen palvelimeen ja editori osaa näyttää MVC-arkkitehtuurin osat oikein. (Zend Studio 8 vs. PDT.)

Vaikkei opinnäytetyö keskittynytkään PHP-kehitykseen, oli Zend Studio luonnollinen valinta kehitysympäristön pohjaksi.

3.2 Aptana Studio

Palvelinohjelmointiin keskittyvänä editorina Zend Studion HTML-, CSS- ja etenkin JavaScript-tuki ovat melko puutteellisia. Näitä puutteita korjaamaan siihen asennettiinkin Aptana-editorin plugin-versio.

Aptana Studio on Zend Studion tapaan Eclipse-pohjainen tuote, mutta ilmainen. Aptana on suunnattu yleisesti www-kehitykseen, myös palvelinohjelmointiin. Zend Studioon verrattuna se ei tarjoa tiettyjä edistyneitä ominaisuuksia, mutta on toiminnaltaan varmempi ja kevyempi.

Opinnäytetyössä Aptanan ominaisuuksista kiinnosti sen tuki JavaScriptillä ja erityisesti erilaisille JavaScript-kirjastoille. Aptanan tekstieditori osaa esimerkiksi jQueryn ominaisuudet ja osaa ehdottaa ja täydentää koodia kehittäjän sitä kirjoittaessa. Myös Aptanan CSS-editori on huomattavasti Zend Studion vastaavaa edistyneempi.

3.3 Firebug

Firebug on www-kehittäjille suunnattu lisäosa Mozilla Firefox -selaimeen. Sen tärkeimpiä ominaisuuksia ovat muun muassa DOM-puun reaaliaikainen tutkiminen ja JavaScript-debuggeri. Firebug tarjoaa myös Firefoxiin monipuolisen konsolin, johon voidaan koodista suorituksen aikana tulostaa merkkijonoja ja muuttujia. Firebugin konsoli osaa esimerkiksi purkaa auki JavaScript-oliot ja taulukot, jolloin niiden sisältöä päästään tutkimaan suorituksen eri vaiheissa.

Firebugin konsoli tarjoaa myös työkaluja sovelluksen suorituskyvyn mittaukseen. Tällä tavalla voidaan mitata aika, joka kuluu tietyn toimenpiteen suorittamiseen ja näin tehdä suorituskykyä parantavia muutoksia. (What is firebug)

3.4 JSLint

JavaScript on perinteisiin olio-ohjelmointiekieliin verrattuna löyhä ja epävakaa kieli. Se on tyypitetty huonosti ja huolimaton kehittäjä voi helposti luoda koodiin bugeja, joiden löytäminen on työlästä. Esimerkiksi muuttujan alustaminen ilman ”var” -etuliitettä tekee muuttujasta globaalin, joka saattaa sopivassa tilanteessa aiheuttaa isoja ongelmia.

JSLint on kehitetty korjaamaan näitä JavaScriptin puutteita. Se etsii koodista erilaisia ongelmakohtia. Osa niistä on selkeitä syntaksivirheitä, joiden lisäksi JSLint tunnistaa syntaksiltaan oikean, mutta muuten kyseenalaisen koodin. (What is JSLint?)

JSLint on toteutettu JavaScriptillä, minkä ansiosta sitä voidaankin käyttää erilaisissa ympäristöissä. Yksinkertaisin vaihtoehto on JSLintin www-sivuilla sijaitseva lomake, mutta ohjelmaa pystyy käyttämään paikallisesti työpöytäkäytössäkin. Opinnäytetyössä JSLint asennettiin Eclipse-lisäosana Zend Studio -editoriin. Lisäosa tarjoaa yksinkertaisen käyttöliittymän, jonka avulla voidaan valita koodin tarkistuksessa käytettävät asetukset. Tarkistus voidaan myös määritellä ajettavaksi aina tiedoston tallennuksen yhteydessä, jolloin tarkastuksesta tulee automaattinen rutiini.

3.5 qUnit

Media Cabinetilla yksikkötestit ovat osa PHP-kehitystä ja niitä käytetäänkin laajasti. Yksikkötestit haluttiin käyttöön myös JavaScript-toteutukseen, vaikkei niistä aiempaa kokemusta ollutkaan. Mitään aiempia tekniikoita tai käytänteitä ei siis ollut aiemmin määritelty.

JavaScriptille on saatavilla useita yksikkötestaukseen tarkoitettua kirjastoja. Suurin osa näistä keskittyy pelkkään yksikkötestaukseen, mutta osa sisältää ominaisuuksia myös www-sovelluksen laajamittaisempaan testaukseen. Opinnäytetyössä päätettiin keskittyä yksikkötestaukseen ja jättää käyttöliittymän muu automaattinen testaus pois.

Saatavilla olevista vaihtoehdoista valittiin käyttöön qUnit. qUnit on jQueryn kehittämä ja käyttämä monipuolinen kirjasto. Se tarjoaa testaukseen yksinkertaiset metodit, joille testattava koodi ja odotettu tulos annetaan parametrina. qUnit-testiympäristön toteutus on lisäksi hyvin nopeaa ja käytettävyys hyvä. Testitapaukset voidaan jakaa moduuleihin ja tarvittaessa kaikkien testien sijaan vain tiettyjen moduulien testit voidaan suorittaa. (qUnit)

```
test("a basic test example", function() {
  ok( true, "this test is fine" );
  var value = "hello";
  equals( "hello", value, "We expect value to be hello");
});
```

Opinnäytetyössä yksikkötestauksesta päätettiin tehdä kiinteä osa kehitysprosessia tekemällä kehitystyö testivetoisesti. Testauksessa käytettävä qUnit teki testitapausten kirjoittamisesta niin helppoa, että tätä pidettiin parhaana tapana kattaa mahdollisimman suuri osa koodista testeillä.

4 KÄYTTÖLIITTYMÄSSÄ KÄYTETTÄVÄT TEKNOLOGIAT

4.1 JavaScript-kirjastot

Opinnäytetyön alussa tavoitteena oli, ettei käyttöliittymäkerrosta sidottaisi mihinkään JavaScript-kirjastoon. Ihanteena oli, että moduulikehittäjä voisi valita muutamista vaihtoehdoista kirjaston, jota myös käyttöliittymäkerroksen komponentin sitten käyttäisivät.

Tämä todettiin kuitenkin lähes mahdottomaksi toteuttaa kirjastojen erilaisten ominaisuuksien vuoksi. Lisäksi se tekisi FOAF UI -komponenttien koodista huomattavasti monimutkaisempaa kirjoittaa ja lukea.

Valittavissa olevista JavaScript-kirjastoista FOAF:n käyttöliittymäkerrokseen valittiin käyttöön jQuery. jQuery on vaihtoehdoista suosituin ja yleisin, jotka olivat hyvä lähtökohta. Jquery on myös ominaisuuksiltaan ja keveydeltään ylivertainen. Erityisesti elementtien hakemiseen DOM-rakenteesta, sekä niiden muokkaamiseen jQuery tarjoaa kilpailijoitaan paremmat ominaisuudet.

Jqueryn ongelmana JavaScript-kirjastoa valitessa nähtiin sen puutteelliset ominaisuudet ison sovelluksen rakentamiseen. Muissa kirjastoissa tähän tarkoitukseen on monipuolisempia ominaisuuksia. Tätä ei kuitenkaan nähty ongelmana FOAF UI -prototyypin yksinkertaisen arkkitehtuurin vuoksi. Suurin osa toiminnallisuudesta tulisi sijaitsemaan jQueryä laajentavina lisäosina.

4.2 HTML5

FOAF on tarkoitus alusta asti rakentaa uusimpien tekniikoiden päälle. Myös HTML:n dokumenttityypiksi haluttiin uusin HTML5, jonka uusia mahdollisuuksia tultaisiin käyttämään.

Uusista ominaisuuksista prototyypin toiminnallisuuden kannalta tärkeimmäksi todettiin niin kutsutut data-attribuutit. HTML:n vanhemmissa versioissa HTML-elementin attribuutit olivat tarkkaan rajoitetut, eikä oikeaoppiseen dokumenttiin saanut määritellä elementeille omia attribuutteja. HTML5:n myötä tämä oli mahdollista ”data-” -etuliitteellä (HTML5: Custom data attributes.) Tämän ominaisuuden varaan tulisi rakentumaan FOAF UI -komponenttien tapa antaa asetuksia HTML-elementissä.

4.3 Jquery-plugin tekniikat

4.3.1 Käyttötarkoitus ja ominaisuudet

jQuery-kirjastoa, toisin kuin monia sen kilpailijoita, ei ole kehitetty modulaariseksi. jQuerystä on pakko ottaa käyttöön koko sen lähdekoodi. jQueryn laajentaminen lisäosilla on kuitenkin yksinkertaista. Näitä lisäosia löytyy valmiina hyvinkin paljon, mutta myös omien tekeminen on helppoa JavaScriptistä tutulla syntaksilla.

Näiden lisäosien suosion takia sitä pidettiinkin parhaana tekniikkana opinnäytetyön pohjaksi. Tällä tavalla toteutettu komponentti on käyttöönotoltaan ja ominaisuuksiltaan loppukäyttäjille jo ennestään tuttu.

jQuery-lisäosien toteuttamiseen on käytänteistä huolimatta monia tapoja. Osa tekniikoista toimii vain yksinkertaisessa käyttötarkoituksessa, jolloin esimerkiksi lisäosan ilmentymiä ei ole sivulla samanaikaisesti enempää kuin yksi. Jotkut lisäosat taas on toteutettu tarkoituksella tai vahingossa ”tilattomiksi”. Tämä tarkoittaa esimerkiksi

sitä, ettei ilmentymä luontinsa jälkeen viittaa DOM-elementtiin, johon se liitetään. Opinnäytetyön jQuery-lisäosien ehdoton vaatimus olikin niiden tilatietoisuus.

Opinnäytetyön jQuery-lisäosien toteutustavan ominaisuuksiksi määriteltiin:

- Mahdollisuus useampiin ilmentymiin yhdellä sivulla
- Vain yksi nimi per lisäosa jQueryn nimiavaruudessa
- Mahdollisuus antaa parametreja oliota luodessa
- Mahdollisuus käyttää olion julkisia metodeita
- Yksityiset metodit käytössä olion sisällä, mutta ei julkisesti sen ulkopuolella

Tutustuttaessa erilaisiin lisäosien toteutustapoihin löydettiin kolme harkitsemisen arvoista tekniikkaa. Nämä kolme ja useat muut tekniikat ja niiden toiminta kokeiltiin yksinkertaisilla esimerkeillä. Nämä kolme valittiin esiteltäväksi niiden vaihtelevien lähestymistapojensa ja erilaisuutensa vuoksi.

4.3.2 jQuery plugin

Yksinkertaisin tapa tehdä oikea jQuery-lisäosa on jQueryn omilla sivuilla esiteltynä. Ohje kertoo askeleittain, miten oma lisäosa tehdään ja mitä ominaisuuksia siihen voidaan liittää. Suurin osa saatavilla olevista jQuery-lisäosista onkin tehty tällä tekniikalla. Siksi saatavilla on hyvin paljon dokumentaatiota ja esimerkkejä. Se on myös paras ja varmin tapa aloittaa oman lisäosan rakentaminen. (jQuery plugin authoring.)

jQueryn oman tekniikan ongelmana on, ettei lisäosalla luotu olio ole tilatietoinen. Esimerkin lisäosa ei toimi odotetulla tavalla tilanteessa, jossa olion DOM-elementtiin on kytketty tapahtumakäsittelyä.

Tilatietoisella tarkoitetaan tässä tilanteessa oliota, jonka metodeita pystyy käyttämään ja ominaisuuksia muuttamaan ilmentymän luonnin jälkeen. (Petersen D. Stateful jQuery plugins with jQuery UI’s widget factory.)

4.3.3 jQuery plugin framework

jQuery plugin framework on Australialaisen jQuery-aktiivin kehittämä tekniikka lisäosan pohjaksi. Tekniikka eroaa jQueryn omasta huomattavasti monimutkaisuudellaan ja lähestymistavaltaan.

Monimutkaisuutensa vuoksi tekniikka todettiin käyttökelpoiseksi, mutta liian monimutkaiseksi FOAF UI -prototyyppiin. Periaatteeltaan ja yksinkertaisten testien perusteella se kuitenkin vaikutti toimivalta.

4.3.4 jQuery UI factory

jQueryn kehittäjät ovat perustaneet jQueryn rinnalle käyttöliittymään keskittyneen jQuery UI -projektin. jQuery UI:n tarkoituksena on tarjota käyttäjälle valmiita komponentteja ja toiminnallisuuksia monipuolisten käyttöliittymien rakentamiseen. jQuery-kirjastoon perustuvana jQuery UI vastaa syntaksiltaan sitä täysin ja on siten helposti omaksuttavissa. (jQuery UI 1.6 – the User Interface Library for jQuery, sivu 9)

jQuery UI:n valmiiden komponenttien pohjana on kirjaston Widget Factory – ominaisuus. Kyse on jQuery UI-olion metodista, jolla komponentit luodaan. jQuery UI komponenttia ei voi suoraan kutsua jQuery-lisäosaksi, mutta se soveltuu siihen hyvin. Widget Factory tarjoaa muun muassa ominaisuudet metodien ja attribuuttien määrittelemiseksi julkiseksi tai yksityiseksi. Metodin luoma lisäosa suorittaa myös automaattisesti joitakin toimintoja, kuten metodikutsuja ja oletusasetusten korvaaminen parametrina annetuilla. Widget factoryllä luotu komponentti voi myös laajentaa olemassa olevaan omaa tai valmista komponenttia. Tämä ominaisuus tarjoaa esimerkiksi perinteisistä olio-ohjelmointikielistä tutun tavan luoda abstrakti komponentti. Komponenttiin voidaan näin lisätä yleisiä eri komponenteissa käytössä olevia ominaisuuksia ja muut komponentin voivat sitten periä nämä ominaisuudet.

```
$.widget('ui.foafMessage', {
        _create: function(params) {
        },
        _init: function(options) {
        },
        publicMethod: function() {
        },
```

```
        _privateMethod: function() {
        },
        _privateAttribute: true,
        options: {},
});
```

jQuery Widget Factoryn huonoin puoli on sen riippuvuus jQuery UI – kirjastosta. Kirjastosta voidaan koota tapauskohtaisesti hyvin kevytkin versio, jolloin loppukäyttäjän selain joutuu lataamaan ylimääräistä toiminnallisuutta mahdollisimman vähän. Widget Factoryn toiminnallisuus voidaan myös kopioida omaksi komponentikseen jQuery UI:n lähdekoodista jos sen käyttöä halutaan ehdottomasti välttää.

Widget Factoryn luoma olio on teknisesti hyvin kevyt eikä sisällä merkittävästi enempää dataa kuin normaali jQuery-olio. DOM-elementtiin liitetty Widget factoryllä luotu komponentti sisältää yhtenä attribuuttinaan normaalin jQuery-olion. Tämän takia se sisältääkin kaikki perinteisen jQuery-lisäosan ominaisuudet ja soveltuu näin tarkoitukseen.

5 KÄYTTÖLIITTYMÄ

5.1 Käyttöliittymän rakenne

Käyttöliittymän arkkitehtuuria lähdettiin suunnittelemaan käytettävyys yhtenä tärkeimmistä tekijöistä. Tehokas ja optimoitu arkkitehtuuri olisi turha, jos se ei tarjoa yksinkertaista rajapintaa loppukäyttäjälle. Oli siis hyvin tärkeää, että käyttäjä tuntee arkkitehtuurin karkealla tasolla ja ymmärtää millaisia vaihtoehtoja hänellä on sen muokkaamiseen ja laajentamiseen.

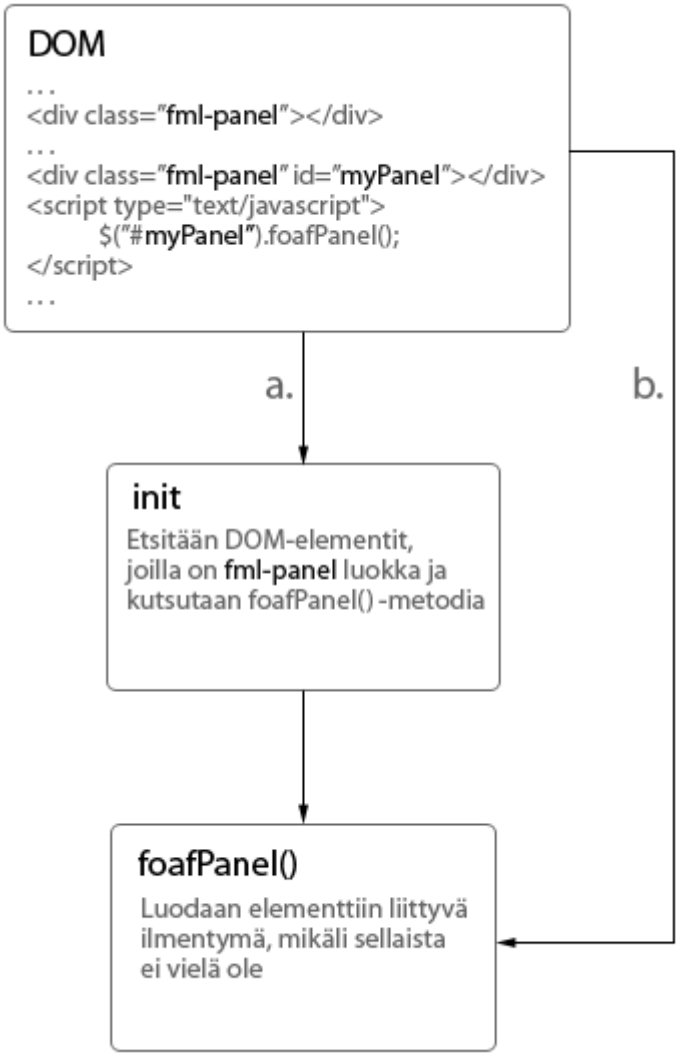
Käyttöliittymän pohjaksi lähdettiin kehittämään oliota per komponentti, jotka toimivat jQuery-kirjaston lisäosina. Tämä mahdollisti jQuerystä tutun syntaksin muun muassa olioiden kytkemisestä DOM-elementteihin ja lisäosan laajentamiseen.

5.2 Instanssien luonti

Käyttöliittymän dynaamisen osan käynnistää init-olio. Olio on jaettu metodeihin sen selkiyttämiseksi, mutta sen toiminta on hyvin yksinkertainen. Kun käyttäjän selain on ladannut sivun HTML:n kokonaan, lisää init-olio FOAF-komponentit DOM-elementeille. Init-olioon on määritelty FOAF-komponentit, joiden perusteella se etsii

DOM-elementtejä, joiden class-attribuutti vastaa määriteltyä. Näin käyttäjä pystyy luomaan FOAF-komponentteja pelkästään lisäämällä HTML-elementille oikean class-attribuutin.

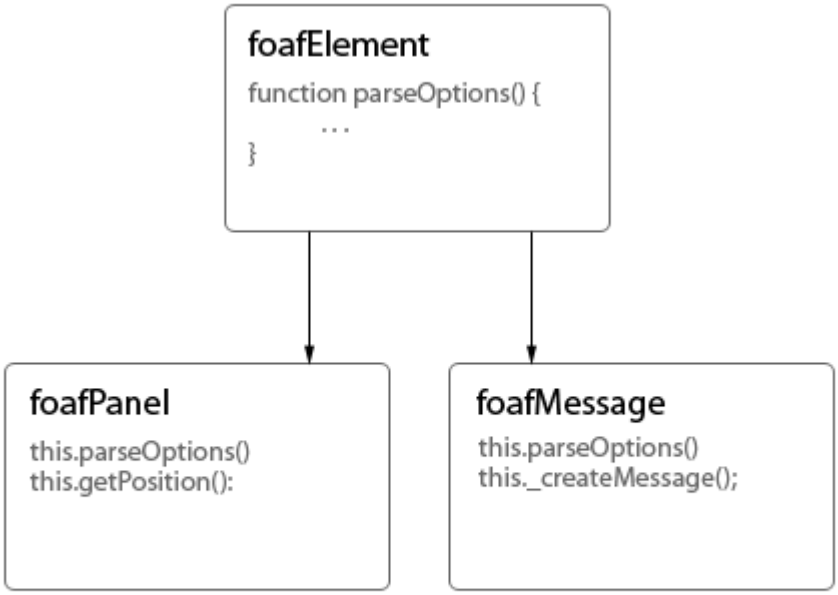
Init-olion kutsun voi halutessaan korvata omalla kutsulla. Kun käyttäjä asettaa FOAF-komponentin DOM-elementille ennen HTML-koodin latauksen ollessa valmis, tunnistaa Init-olio automaattisesti elementille asetetun komponentin ja ohittaa sen (Ks. kuvio 2.). Näin käyttäjä voi antaa komponentin oliolle haluamansa parametrit ja ylikirjoittaa oletuskutsun.



KUVIO 2. Instanssien automaattinen luonti ja tämän ohittaminen

5.3 Abstrakti element-komponentti

Käyttöliittymän komponentit sisältävät osittain keskenään hyvin samankaltaisia ominaisuuksia. Päällekkäisen koodin minimoimiseksi komponenttien pohjaksi tehtiin näennäisesti abstrakti Element-komponentti, jota muut laajentavat. Näin komponenttien yhtenevät ominaisuudet pystyttiin keräämään samaan paikkaan ja määrittelemään vain kerran (Ks. kuvio 3.).



KUVIO 3. FOAF-elementtien ominaisuuksien periytyminen

Tärkein tällainen ominaisuus oli komponentille annettavien asetusten käsittely ja tallennus komponentille. Kaikille FOAF UI:n komponenteille pystytään siis antamaan parametreja monipuolisesti, mutta keskenään yhtenevällä tavalla.

Asetuksia voi antaa komponentille kolmella eri tavalla:

- Parametrina JSON-formaatissa lisätessä oliota DOM-elementille
- DOM-elementin data-fml -attribuutissa JSON-formaatissa

- Merkkijonona DOM-elementin data-fml-X -attribuutteina niin, että X on avain

Näistä ensimmäinen tulee oletuksena init-metodilta ja on tyhjä. Käyttäjän ylikirjoittaessa komponentin kutsun, ylittää se mahdolliset parametrit DOM-attribuutteina annetut asetukset.

Asetusten mahdollisen validoinnin jälkeen ne tallennetaan komponentin options-attribuutiksi. Tämän jälkeen kaikilta FOAF-komponenteilta voi kysyä sen asetuksia option-metodilla. Osa tästä toiminnallisuudesta periytyy suoraan jQuery UI:n Widget Factory-oliolta, mutta osa asetusten läpikäynnistä on toteutettu omaan FOAF Element-komponenttiin

5.4 Komponenttien yhteiset ominaisuudet

5.4.1 Callbackit

Kaikki FOAF-komponentit tukevat callback-olioita, joita kehittäjä voi niille parametrina antaa. Callback-olioiden perustana on custom eventit, joita komponentit laukaisevat. Esimerkiksi Panel-komponentti lähettää ”onUpdate”-eventin, kun se on onnistuneesti ladannut asynkronisesti sisältönsä. Näin kehittäjä voi määritellä vaikka jokaiselle Panel-ilmentymällä erilaisen toiminnallisuuden tähän tilanteeseen.

Jokaisella FOAF UI -komponentilla on omat eventit, joita se kutsuu. Joissakin tapauksissa komponentin eventille on jo asetettu oletustoiminnallisuus, jonka kehittäjä voi ylikirjoittaa.

5.4.2 Elementtien luonti dynaamisesti

Yleisin tapa luoda FOAF UI -komponenttien ilmentymiä on liittää se elementtiin, joka on DOM-rakenteessa jo valmiiksi. jQueryssä on kuitenkin tekniikka, jonka avulla yhdellä koodirivillä voidaan elementti ensin lisätä näkymään ja välittömästi luoda ilmentymä. Tätä tekniikkaa käyttämällä esimerkiksi Message-komponentin HTML voidaan lisätä näkymään samalla, kun itse olio luodaan.

5.5 Grid

Grid-komponentti on käyttöliittymän keskeisimpiä osia. Sen pohjana on näkymään luotu HTML-elementti, jonka tilalle grid-komponentti luodaan. Annettujen asetusten

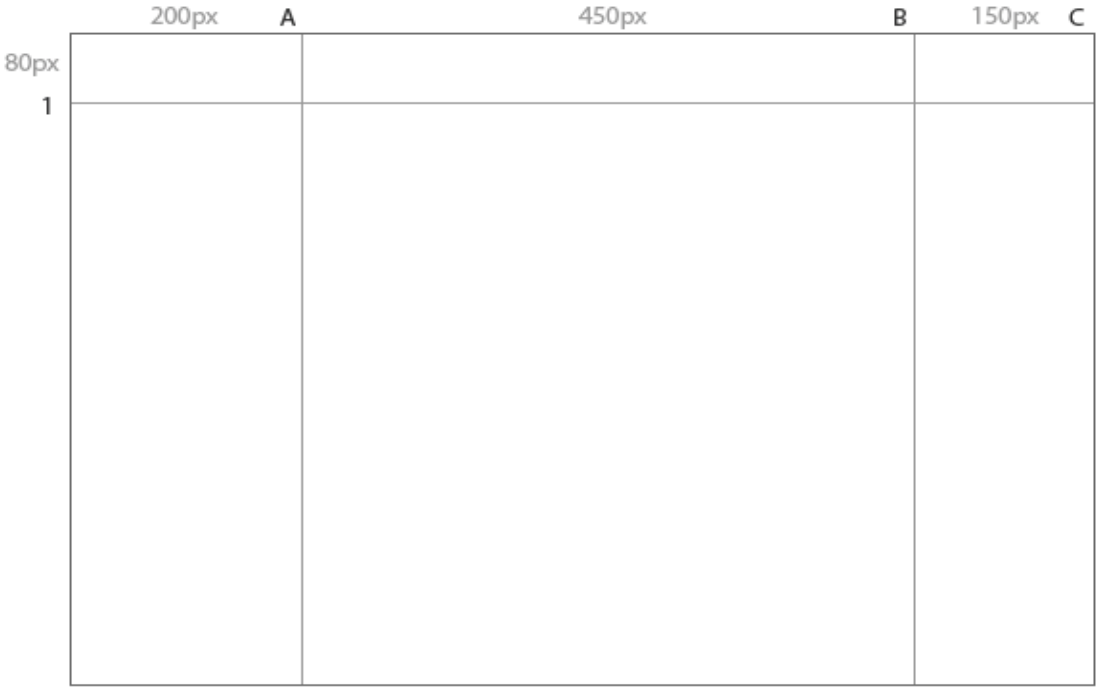
mukaan Gridille luodaan virtuaaliset sarakkeet ja rivit, joihin HTML-elementin sisällä sijaitsevat käyttöliittymän komponentit voidaan asemoida.

Grid-komponentin sarakkeet ja rivit voidaan luoda käyttäen yksiköinä pikseleitä tai prosentteja. Käytettäessä prosentteja komponentti muuntaa itsenäisesti prosentit pikseleiksi, joita sitten käytetään. Gridin koon muuttuminen selainikkunaa suurentaessa tai pienentäessä aiheuttaa prosentuaalisesti määriteltyjen rivien ja sarakkeiden uudelleen laskemisen. Näin pystytään rakentamaan erikokoisille resoluutioille skaalautuva käyttöliittymä.

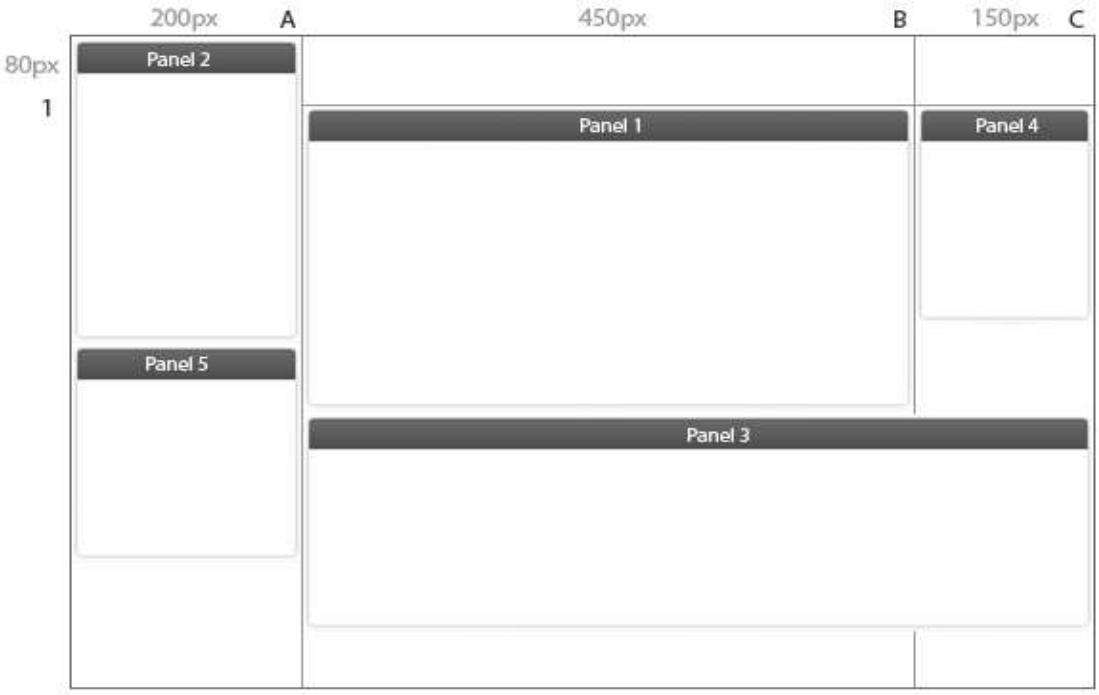
Käytännön esimerkkinä Gridillä voi luoda ulkoasun rungon (Ks. kuvio 4.) palstoineen ja asemoida esimerkiksi Panel-komponentteja oikeaan ja vasempaan reunaan tai yläkulmiin joustavasti (Ks. kuvio 5.). Gridiä ei kuitenkaan ole pakko käyttää ja kehittäjä voi asetella käyttöliittymän komponentit haluamallaan tekniikalla. Muut komponentit eivät ole oletuksena riippuvaisia Grid-komponentista.

Esimerkki Grid-komponentista:

```
<div class="fml-grid" id="layoutGrid" data-fml-
columns="{'a':'0','b':'250','c':'600'}" data-fml-
rows="{'1':'0','2':'100','3':'500'}">
</div>
```



KUVIO 4. Grid-elementillä luotu tyhjä ruudukko



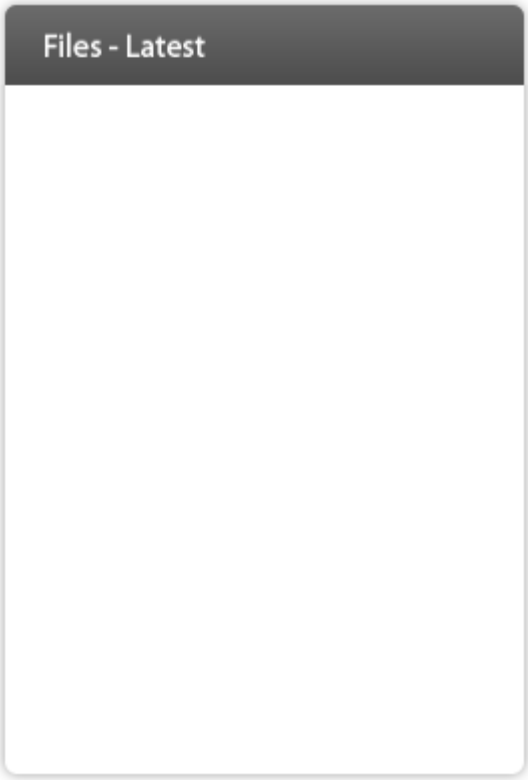
KUVIO 5. Grid-elementillä luotuun taulukkoon asemoitut Panel-oliot

5.6 Panel

Panel-komponentti on FOAF-käyttöliittymäkerroksen ja samalla opinnäytetyön keskeisin osa. Sen tarkoituksena on toimia yleisesti yksikköinä, joihin sisältö jaetaan (Ks. kuvio 6.). Tämän jälkeen Panel-komponentit voidaan asemoida Grid-komponentille määriteltyihin riveihin ja sarakkeisiin.

Esimerkki Panel-komponentin HTML:stä:

```
    <div class="fml-panel" id="fml-panel-1" data-fml-
anchors="{'lt':'b,1','rt':'c,3','lb':'b,3'}" data-fml-
src="/ajax/index.php" data-fml-async="true">
        <h3 class="fml-panel-label">Tools</h3>
        <a href="/crm/customer/new" class="fml-
button">New customer account</a>
    </div>
```



KUVIO 6. Panel-komponentilla luotu olio

Panel tarjoaa oletuksena monipuolisia ominaisuuksia, joiden avulla sen sisältöä voidaan hallita:

5.6.1 Sisällön lataus

Panel-komponentin parametrilla pystytään määrittelemään lähde, josta sen sisältö ladataan. Sisällön lataamiseen on kaksi eri tekniikkaa, joita käyttämällä asynkronisesti tai FOAF:n omaa protokollaa käyttäen.

Ajax-tekniikkaa käyttäessä pyyntö tehdään jQueryn Ajax-kirjastoa apuna käyttäen. Ennen pyyntöä Panelin HTML-elementin sisällön loppuun luodaan valmiiksi DIV-elementti, johon pyynnön vastaksena saama näkymä asetetaan. Palvelimen pyyntöön lähettämässä vastauksessa on mahdollista lähettää varsinaisen sisällön lisäksi myös headerina linkkejä CSS- tai Javascript tiedostoihin. Panel elementti käy nämä läpi ja asettaa ne näkymään käyttöön. Tämä mahdollistaa ulkoisten komponenttien latauksen niin, että ne voidaan ottaa osaksi käyttöliittymää kokonaisuutena.

Panel-komponentille, jonka sisältö ladataan automaattisesti, on myös mahdollista määritellä sen päivitystapa. Panel voidaan päivittää vain kerran, tai interval-asetusta

käyttämällä määritellä päivitysten aikavälin. Sen lisäksi, että interval-asetus annetaan alussa, on sitä mahdollista vaihtaa myös käytön aikana. Päivitys voidaan ottaa pois käytöstä tai vaihtaa kokonaan kohdetta, josta sisältö ladataan.

Vaihtoehtona tutulle Ajax-tekniikalle on FOAF:n oma protokolla, joka tekee latauksen suoraan palvelimella. FOAF:n oma protokolla ottaa parametrinaan JSON-muotoisen olion, jossa on määritelty moduuli, controlleri ja action, jolta sisältö ladataan.

5.6.2 Ylimääräiset elementit

Käytännön tapauksissa vastaan tulee tilanteita, joissa HTML-elementille olisi esimerkiksi saatava taustakuva sen jokaiseen nurkkaan. Tämän ollessa vasta CSS3-version ominaisuus, on kehittäjien ollut pakko käyttää ylimääräisiä HTML-elementtejä ratkaistakseen ongelma CSS2-yhteensopivalla tavalla. Yksinkertaisinta on luoda HTML-elementille kolme lapsielementtiä ja sijoittaa varsinainen sisältö tämän rakenteen sisään. Näin jokaiselle elementillä pystytään asettamaan omat CSS-tyylinsä.

Panel-komponentin koostuessa yhdestä DIV-tyyppisestä HTML-elementistä, haluttiin siihen luoda automaattisesti ominaisuus tällaisia tilanteita varten. Käyttäjä voi yksinkertaisella asetuksella määritellä luodaanko Panel-komponentille nämä elementit. Panelin ulkoasun muokkaus jopa tapauskohtaista on näin helppoa.

5.6.3 Otsikko

Panel-komponentille haluttiin luoda käyttöjärjestelmistä tuttu tapa esittää otsikko. Otsikko kuvaisi Panel-komponentin sisältöä ja toimisi kiintopisteenä joillekin ominaisuuksille. Vaikka Panel olisikin tyhjä, olisi siinä vähintään yläreunan otsikko. Oletuksena otsikko luodaankin tyhjänä, jos komponentille ei ole label-asetusta annettu. Oletusasetuksen voi kuitenkin vaihtaa ja näin estää tyhjänkin otsikon näyttämisen.

5.7 Message

Message-komponentti on yksinkertainen erilaisten viestien ilmoittamiseen tarkoitettu komponentti. Sen avulla pystytään esittämään virhe-, varoitus- ja infoviestejä käyttäjälle.

Message-komponenttia voidaan käyttää kahdella tavalla. HTML-koodiin voidaan tehdä elementti viesteineen, josta automaattisesti tehdään Message-komponentin il-

mentymä. Näin palvelinohjelmoinnilla voidaan tulostaa suoraan esimeriksi vuorovaikutteisia virheilmoituksia. Toinen vaihtoehto on luoda ilmentymä JavaScript koodista HTML:stä riippumatta. Tätä tekniikkaa voivat käyttää muut FOAF-komponentit virheilmoitustensa tulostukseen.

Message-komponentin toimintaa on mahdollista muuttaa asetuksilla, kuten muitakin FOAF UI -komponentetteja. Oletuksena luotuun viestielementtiin lisätään ikoni, josta viesti poistetaan näkymästä. Poisto tehdään poistamalla DOM-elementti ja komponentin ilmentymä. Kehittäjä voi kuitenkin callback-funktion määrittelyllä muuttaa tämän ikonin toimintaa ja esimerkiksi poiston sijaan vain piilottaa sen.

Message-komponentilla on neljä tyyppiä, jotka kuvaavat yleisimpiä käyttäjälle näytettäviä viestityyppejä:

- Tiedote
- Varoitus
- Virhe
- Onnistuminen

5.7.1 Dialog

Dialog-komponentin on tarkoitus toimia esimerkkinä komponentista, joka sisältää mahdollisimman vähän omaa koodia ja käyttää toiminnallisuudessaan valmista toteutusta. Dialog-komponentin ydin on jQuery UI:n dialog-widget, jolle tarjotaan ominaisuuksiltaan Message-komponenttia muistuttava käyttöliittymä.

Dialog vastaa käyttötavaltaan Message-komponenttia. Myös sitä pystyy käyttämään joko lisäämällä oikean HTML-elementin koodiin tai luomalla instanssin dynaamisesti JavaScript-koodista.

6 POHDINTA

6.1 PHP-parseri

Opinnäytetyön alussa suunnitelmana oli, että JavaScript-toiminnallisuuden lisäksi käyttöliittymäkerroksen arkkitehtuuriin kuuluisi PHP-osa. Tarkoituksena oli, että kehittäjä kirjoitti HTML-koodin sijaan yksinkertaisia XML-elementtejä. Elementit oli eroteltu omaan nimiavaruuteensa, johon kuuluvat elementit ohjelma olettaa FOAF-elementeiksi ja käy läpi. Parseri tarkistaa mahdolliset parametrit ja tulostaa ne HTML-muotoon.

Parserin kehittämiseen ja erilaisten toteutustapojen vertailuun kulutettiin opinnäytetyön alussa paljon resursseja. PHP tarjoaa HTML-koodin läpikäymiseen useita erilaisia mahdollisuuksia, kuten SimpleXML, Xpath, DOM-luokat ja säännölliset lauseet. Tekniikoista kaikki viimeistä lukuun ottamatta käsittelisivät sisältöä XML-elementteinä ja olisivat näin helposti alttiina virheille. Säännöllisiä lauseita käyttämällä HTML-koodia käsiteltäisiin merkkijonona, joka sietäisi koodin mahdollisia virheitä suhteellisen hyvin. Suorituskyky- ja muiden pienten ongelmien johdosta tämä vaihtoehto kuitenkin hylättiin.

Lopulliseksi toteutustavaksi valittiin PHP:n DOM-luokat. Näkymästä luodaan DOM-Document-olio, jonka DOMNode-lapsielementit käydään läpi. Määriteltyyn FOAF-nimiavaruuteen kuuluvat elementit käydään rekursiivisesti läpi. Löydetyistä elementeistä luodaan niiden class-attribuutin perusteella olioita. Jokaisella FOAF UI -elementillä on oma luokkansa, joka määrittelee mitä ja miten se tulostetaan. Näiden luokkien pohjana on abstrakti Element-luokka, jolle oli kerätty komponenttien yhteiset ominaisuudet.

Jo PHP-parserin suunnitteluvaiheessa tiedettiin parsinnan olevan raskasta. FOAF-elementit sijaitsivat yleensä sisäkkäin ja sisälsivät monia asetuksia attribuutteina, jolloin realistiset käytännön esimerkit olivat raskaimmasta päästä. Tämän ongelman korjaamiseksi PHP-parseriin suunniteltiin alusta asti kattava välimuisti, jonka avulla iso osa parsimisesta pystyttiin jättämään väliin ja käyttäjälle palauttamaan näkymä välimuistista.

PHP-parserin ollessa toimintakunnossa päätettiin sen käytöstä luopua. Syynä tähän oli sen monimutkaisuus suhteessa hyötyyn. Yksinkertaisten XML-elementtien ongelmana oli niin kutsuttu uusi kieli, jonka kehittäjä joutuisi opettelemaan sitä käyttääkseen. Vaihtoehtona tälle oli kirjoittaa suoraan sopivaa HTML-koodia, mikä teki prosessista huomattavasti yksinkertaisemman. PHP-parserin mahdollisuutta ei unohdettu täysin. Sitä ei kuitenkaan haluttu prototyyppiin mukaan.

6.2 jQuery-plugin

PHP-parserin käytöstä luopumisen jälkeen käyttöliittymä perustui pelkkään selainpuolen tekniikoihin. Tämä muutti merkittävästi esimerkiksi tapaa, jolla asetuksia annettiin FOAF-komponenteille. PHP-parserin tapauksessa JavaScript-kutsut ja ilmentymien luonnit voitiin generoida XML:stä, eikä kehittäjän tarvinnut välittää niistä. Asetusten antaminen parametrina onnistuttiin kuitenkin toteuttamaan täysin suunnitellulla tavalla.

jQuery UI:n Widget Factory -ominaisuus toimi toteutusvaiheessa odotetulla tavalla. Komponenttien kyky laajentaa toisiaan osoittautui tarpeelliseksi ominaisuudeksi. Käyttöliittymäkerroksen prototyypin jatkokehityksessä periytymistä tullaan varmasti muuttamaan. Komponenttien lukumäärän lisääntyessä Element-komponentin annetut asetukset läpi käyvä toiminnallisuus tulisi erottaa komponentille, jolta Element ne perisi. Tämä mahdollistaisi entistä paremmin käyttöliittymän komponenttien jakamisen kahteen ryhmään: Visuaalisiin asemoitaviin ja näkymättömämpiin komponentteihin.

6.3 Kolmannen osapuolen kirjastot

6.3.1 Käytetyt jQuery-lisäosat

Käyttöliittymäkerroksen toteutusvaiheen aikana todettiin, että joiltakin osin jQuery ei tarjonnut toivottuja ominaisuuksia. Näitä tilanteita korjaamaan valittiin muutamia erillisiä JavaScript-kirjastoja. Kirjastot olivat hyvin kevyitä ja tiettyyn tarkoitukseen kehitettyjä, joiden ansiosta niiden käyttöä pidettiin perusteltuna.

6.3.2 jQuery.Timers

Yleinen tilanne JavaScript-sovellusten kanssa on tarve käyttää ajastimia. Ajastimilla tiettyjä toiminnallisuuksia voidaan esimerkiksi suorittaa viiveellä tai toistuvasti tietyillä väliajoilla. JavaScriptissä tämä onnistuu kielen omalla setTimeout-funktiolla, jonka käyttö ei aina ole kovin suoraviivaista.

jQuerylle on kehitetty lisäosa korjaamaan tilannetta helpottamaan ajastimien käyttöä. Lisäosan avulla ajastin liitetään yleensä tiettyyn HTML-elementtiin ja rekisteröidään globaaliin olioon. Näin ajastimeen päästään ilmentymän luonnin jälkeen käsiksi HTML-elementin metodia kutsumalla. Tämä mahdollistaa esimerkiksi ajastimen pysäytyksen.

Käyttöliittymäkerroksessa ajastin-lisäosaa käytettiin Panel-komponenttien toiminnallisuudessa, jolla haettiin Panelin sisältöä asynkronisesti. Sisältö oli mahdollista hakea ajastetusti säännöllisin väliajoin, joka mahdollisti Panelin sisällön huomaamattoman päivityksen.

Ajastin-lisäosa mahdollisti helpon tavan hallita Panel-kohtaisesti sen päivitystapaa. Ajastettu päivitys pystyttiin käytön aikana esimerkiksi ottamalla pois kutsumalla Panelin stopAjax-metodia.

6.3.3 underscore.js

Käytetty underscore.js on pieni ja kevyt JavaScript-kirjasto, jonka on tarkoitus helpottaa pieniä ja yksinkertaisia operaatioita. Tärkeimpiä ominaisuuksia oli taulukoiden, olioiden ja merkkijonojen käsittely, joihin kirjasto tarjoaa yksinkertaisia apuvälineitä. Nämä ominaisuudet olivat paikoitellen päällekkäisiä jQueryn omien ominaisuuksien kanssa, mutta tästä huolimatta underscore.js otettiin käyttöön.

6.4 Testivetoinen kehitys

Testivetoinen kehitys ja yksikkötestaus olivat ehdottomasti opinnäytetyön arvokkaimpia kokemuksia. Monissa tapauksissa testien kirjoittaminen koettiin työlääksi taakaksi, mutta testit osoittivat hyötynsä usein. Opinnäytetyön ohjelmointijakson aikana jotkut ominaisuudet kehitettiin uudelleen muutamaan kertaan. Nämä ominaisuudet liittyivät muihin toiminnallisuuksiin, ja näin muutokset vaikuttivat laajalti.

Laadun ja toiminnan valvominen ilman yksikkötestausta ja valmiita testitapauksia olisi hyvin työlästä, ellei mahdotonta. Valmiiden testitapausten avulla oli helppo huolehtia sovelluksen toimivan odotetun mukaisesti.

Yksikkötestien testitapaukset toimivat myös hyvin dokumentaationa. Yksittäisen metodin testitapauksista on helppo päätellä, mitä sen tulisi palauttaa määritellyillä parametreilla.

Paras esimerkki opinnäytetyössä yksikkötestauksen hyödystä on toiminnallisuus, joka määritteli annettujen parametrien perusteella Panel-komponentin paikan Gridissä. Grid-komponentille määriteltiin rivit ja sarakkeet, joihin komponentteja asemoitiin. Suunnitteluvaiheen aikana määritellyt säännöt mahdollisti suuren määrän erilaisia yhdistelmiä asemointiin, mikä teki kattavasta testauksesta työlästä.

Elementin paikan Gridissä laskeva toiminnallisuus toteutettiin omana metodinaan, joka ei tehnyt elementille vielä mitään. Toteutuksesta tehtiin hieman monimutkaisempi, jolloin metodi saatiin palauttamaan testattavan olion. Olio sisälsi asemoitavan elementin mitat ja sijainnit, joiden tarkistaminen testitapauksilla oli helppoa. Tätä yksittäistä ominaisuutta testaamaan tehtiin 15 testitapausta.

Yksikkötestauksesta tulee varmasti osa FOAF UI:n tulevaa kehitystä. Niin paljon se helpottaa monimutkaisten ominaisuuksien testausta ja laadun tarkkailua.

6.5 Prototyypin tulevaisuus

Käyttöliittymäkerroksen valmistuessa otettiin se käyttöön FOAF-prototyyppiin. Kaikkia toteutettuja komponentteja ei ollut tarkoitus käyttää sellaisenaan, mutta esimerkiksi Grid- ja Panel-komponentit pyritään pitämään osana prototyyppiä alusta asti.

Osa opinnäytetyön suunnitteluvaiheen aikana tehdyistä linjauksista muuttui toteutuksen aikana. Osa ominaisuuksista tulee todennäköisesti muuttumaan opinnäytetyön valmistumisen jälkeen. Tärkeintä oli kuitenkin saada tehty tutkimustyö ja prototyyppi käyttöön mahdollisimman pian.

Käyttöliittymä tulee jatkamaan kehitystään osana muuta FOAF-prototyyppiä. Kehitystyö tulee jatkumaan Media Cabinetin toimesta ja on toivottavasti isona osana ensimmäistä julkaisuversiota.

LÄHTEET

HTML5: Custom data attributes. Viitattu 1.4.2011. http://www.w3.org/TR/html5/elements.html#embedding-custom-non-visible-data-with-the-data-attributes

jQuery UI 1.8 Widget Factory. Viitattu 1.4.2011. http://blog.petersendidit.com/post/jquery-ui-1-8-widget-factory

Onko julkaisujärjestelmien aika ohitse? Vastaus: Ei. Viitattu 1.4.2011. http://vierityspalkki.fi/2011/03/28/onko-julkaisujarjestelmien-aika-ohitse-vastaus-ei/

Petersen D. Stateful jQuery plugins with jQuery UI's widget factory. Viitattu 1.4.2011. http://blog.petersendidit.com/post/stateful-jquery-plugins-with-jquery-uis-widget-factory/

qUnit. Viitattu 1.4.2011. http://docs.jquery.com/Qunit

Test-Driven JavaScript Development. Viitattu 1.4.2011. http://tddjs.com/

Unit testing. Viitattu 1.4.2011. http://en.wikipedia.org/wiki/Unit_testing

What is Firebug. Viitattu 1.4.2011. http://getfirebug.com/

What is JSLint?. Viitattu 1.4.2011. http://www.jslint.com/lint.html

Widget Factory. Viitattu 1.4.2011. http://wiki.jqueryui.com/w/page/12138135/Widget-factory

Zend Studio 8 vs. PDT. Viitattu 1.4.2011. http://www.zend.com/en/products/studio/comparison